1. 리액트 프로젝트 생성 - 도스 명령어 입력
2. code 소스 편집 - terminal 메뉴 명령어
3. 리액트 구조
4. 리액트 조건문, 반복문, 변수, css, 이벤트명
5. 리액트 여러 html 태그 조합 1 개 사용자 태그+이벤트처리+css =
리액트 컴포넌트(대문자시작)

class A extends
React.Component
{
render(){
…
return(jsx 문법);
}
ecma6
	function A ()
	{
	…
	return(jsx 문법);
	}
	ecma5
		const A = ()=>{
		
		…
		return(jsx 문법);
		
		}
		
		ecma6

- 리액트 HOOKS
6. 리액트 컴포넌드간에 데이터 전달 - props 데이터 변경 화면 re-rendering x
7. state - 리액트 컴포넌트 이벤트 처리 데이터값 변경 - 화면 re-rendering o

- staete 내부 변경 - 화면 렌더링-해당클래스
-함수 제공 - 다른 클래스 props 전달

BoardList - getter/setter	1. state - board 배열 2. state - addBoard 함수
<table> <BoardHeading /> <BoardRow />-반복 <BoardWriteForm add={ addBoard } /> - 글쓴 내용	

- 프론트엔드 react + 백엔드 spring boot + mysql

http://localhost:8080 + http://localhost:8080 + localhost:3306

http://localhost:3000 + http://localhost:8080 + localhost:3306

같은 서버인가 판단
http:ip:port ==> 동일하다면 같은 서버

http://localhost:3000 <==> (통신) <==> http://localhost:8080 + localhost:3306

컨트롤러 - service, dao, view-jsp 전달

모델앤뷰로 리턴할때는 addobject 로 결과값을 여러개 줄 수 있었는데 리액트로 할 경우에는 불가능한 걸까요?

server.js
http://localhost:3000
1> 서버가 다르면 비동기통신 허용 x
2> "가짜"
설정
package.json
"proxy" : "http://localhost:8081",
 	
class Connection .....{
render(){
$.ajax({url : /login'});
axios.get('/helloreact', .. method:
data ...)
결과받아서 state board 저장
}
 	
return (..{this.state.board}...)
}
	리액트 연동 컨트롤러
	@RestController
	class A {
	@RqquestMapping("/login")
	@ResponseBody
	public Map/List/[] a(){
	return xx:
	}
	
	@RqquestMapping("/login")
	@ResponseBody
	public xxx a (){
	return "login";
	}
	
	@RqquestMapping("/login")
	@ResponseBody
	public xxxx a (){... }

8. 서버 통신
9. 리액트 컴포넌트 연결 - router(route - 길)
http://localhost:3000/index.html

*.js 파일 정의

-

axios
javascript 어플리케이션에서 서버 통신을 하기 위한 HTTP 통신 라이브러리다.
react 에서도 사용 가능하다.

설치
npm add axios

Axios HTTP 요청 메서드 종류

axios.get(url , config{params })	서버에서 데이터를 가져올 때 사용하는 메서드입니다. 두 번째 파라미터 config 객체에는 헤더(header), 응답 초과시간 (timeout), 인자 값(params) 등의 요청 값을 같이 넘길 수 있습니다.
axios.post(url, data, config)	서버에 데이터를 새로 생성할 때 사용하는 메서드입니다. 두 번째 파라미터 data 에 생성할 데이터를 넘깁니다.
Axios HTTP 요청 config 옵션	method : method 는 요청 메서드 종류. 기본값 get. url : 서버 URL. baseURL : baseURL 은 액시오스 인스턴스를 생성할 때, 인스턴스의 기본 URL 값을 정할 수 있는 속성. 보통 API 서버의 기본 도메인을 설정하고, 인스턴스 별로 URL 을 뒤에 추가하여 사용. headers : 헤더값 수정. params : 요청에 붙일 URL 파라미터. data : data 는 HTTP 요청시 전송 데이터. responseType : 서버의 응답 데이터 형식. arraybuffer, document, json, text, stream. 기본 값은 json

예>

react	spring controller

axios.post('http://localhost:8081/helloajax')
.then(function(response) {
console.log(response.data);
});
	@CrossOrigin(origins="*")
	@RequestMapping("/helloajax")
	@ResponseBody
	public MemberDTO test() {
	return new MemberDTO("ID","PASSWORD");//json 자동변환(스프링부트내장)
	}

axios.get('http://localhost:8081/helloajaxparam', {	@CrossOrigin(origins="*") @GetMapping("/helloajaxparam")

params: { id: 'get 방식호출 vue', password:100 }, headers: { 'Content-Type': 'application/json' } }) .then(function(response) { console.log(response.data); });	@ResponseBody public MemberDTO test(String id , int password) { return new MemberDTO(id,"PASSWORD"+password);//json 자동변환(스프링부트내장) }
axios.post('http://localhost:8081/helloajaxparam', {id:"vue axios", password:'100'})) .then(function(response) { console.log(response.status); console.log(response.data); });	// //주의할 점으로 GET 통신에서는 @RequestParam 을 사용하지만, POST 통신에서는 @RequestBody 를 사용한다. @CrossOrigin(origins="*") @PostMapping("/helloajaxparam") @ResponseBody public MemberDTO test2(@RequestBody LoginDTO dto) { return new MemberDTO(dto.getId(),"PASSWORD"+dto.getPassword());//json 자동변환(스프링부트내장) } } class LoginDTO{ String id; int password; public String getId() { return id; } public void setId(String id) { this.id = id; } public int getPassword() { return password; } public void setPassword(int password) { this.password = password; } }

6-3 CORS 문제
HTTP 프로토콜은 URL 구조 가운데
프로토콜명,ip,port 동일시 같은 서버, 아니면 다른 서버로 간주한다.

CORS(Cross Origin Resource Sharing)란 도메인과 포트가 다른 서버로 client 가 요청했을때 브라우저가 보안상 이유로 API 를 차단하는 문제이다.

예를들어 client 는 7070 포트에서 동작하고 server 는 8080 포트에서 동작할 경우나 또는 **로컬에서 다른 서버로 호출할때** 발생한다.

즉, 간단하게 동일출처 정책으로 같은 도메인이 아니면 받는 server 에서 모든 client 요청에 대한 cross-origin HTTP 를 허가해주지 않는다면 브라우저단에서 위와 같은 cors 에러가 발생한다.

해결책은
1.서버
:요청받는 server 에서 모든 요청을 허가한다던지 백단에서 cors 패키지를 설치해서 미들웨어로 처리를 한다던지의 방법으로는 해결할 수가 없다.

스프링 부트의 경우
```
@CrossOrigin(origins="*")
@RequestMapping("/helloajax")
@ResponseBody
public MemberDTO test() {
    return new MemberDTO("ID","PASSWORD");
}
```

또는

2.클라이언트

2-1 : react 의 경우, 프로젝트의 루트 디렉토리에 package.json 라는 파일에 아래의 코드를 추가한다. (루트 도메인까지 proxy 로 작성한다)

"proxy" : "http:/localhost:8080"

2-2 : 현재 실행을 중단하고 npm start 실행한다.

2-3. server 에 요청하는 url 을 다음와 같이 수정한다. (루트 도메인을 제외한 나머지 url 값을 작성한다)

axios.get(`/helloajax?id=react`)

- router v6(v5 문법 다르다)
설치
npm add react-router-dom

index.html	index.js	xxx.js
.. <div id="root" .....> 	소스 수정 컴파일 실행 <A /> <B /> <C />	

<a ==> 다른 html 파일 이동 링크

<Link ==> index.html 파일 내부 리액트 컴포넌트 이동 링크

SPA – SINGLE PAGE APPLICATION
: 한개의 페이지로 구성된 어플리케이션.
index.html 1개로 모든 react component들을 표현한다.

1>전통적인 웹요청은 사용자가 다른 페이지로 이동할 때 새로운 html 파일을 생성 – 해석한 뒤 브라우저에 보여준다.
2>한 페이지에 많은 정보가 포함된 경우 새 화면으로 이동할 때마다 페이지를 새롭게 로드한다.
3>페이지의 특정 부분만 변경되고 모든 정보를 새로 로드할 필요는 없다 해도 이동할 때마다 새로운 페이지를 로드한다.
4> 네트워크 부하가 크다는 단점이 있다.
5> 해결책은 뷰 렌더링은 브라우저가 담당하고 사용자의 동작이벤트 발생시 필요한 부분만 자바스크립트 이용 업데이트할 수 있는 방법이 있다.
6>새로운 데이터가 필요하면 서버로부터 새로운 데이터만 불러온다.
7> 5+6 은 웹페이지 전체에 변화가 있을 때마다 새로 그리는 것이 아니라 변화가 있는 부분만 갱신한다는 의미이다.
8> 1개의 페이지에서 모든 정보를 보여주는 효율적 방법이다.-spa
9> 1개의 웹페이지를 사용하여 url 구분에 따른 서로 다른 주소에 다른 화면을 보여 주는 것을 라우팅이라고 한다.

예
index.html

```
...
<div id="root"></div>
```

index.js

http://localhost:3000/a	http://localhost:3000/b	http://localhost:3000/c

ReactDOM.render(<A />, document .getElementById ("root"));	ReactDOM.render(<B />, document .getElementById ("root"));	ReactDOM.render(<C />, document .getElementById ("root"));

- 실습

1>react-router-dom 설치

1-1.명령 프롬프트 열고

1-2.npm install -g react-router-dom

2> 새로운 리액트 프로젝트 생성

2-1. cd C:₩kdt-venture₩workspace_react

2-2. create-react-app route-react

2-3. cd route-react

2-4. npm start

2-5. vs code 실행- 폴더열기

3>index.js에서 ,BrowserRouter 컴포넌트 사용

- BrowserRouter : HTML5를 지원하는 브라우저의 url를 감지한다.

4>App.js에서 Route 컴포넌트 사용

router v5

<Route path="uri(주소)" component={보여줄 컴포넌트이름} />

<Route path="uri(주소)" component={보여줄 컴포넌트이름} exact={true}/>

router v6

<Routes>

<Route path='/' element={<Main />} />

<Route path='/boardlist' element={<BoardList />} />

{/* /board로 시작하는 모든 url 처리 . 현재는 1번 게시물만 보여줌 */}

<Route path="/board/*" element={<Board seq="1" />} />
{/* 위 라우트 규칙에 일치하는 라우트가 없는 경우 다음 처리 */}
<Route path="*" element={<NoMatch />} />
</Routes>

5> 파일을 작성하여 실행하고 결과 확인

localhost:3000 or localhost:3000/ ==> Main 컴포넌트

localhost:3000/boardlist ==> BoardList 컴포넌트

localhost:3000/board/1 ==> Board 컴포넌트

localhost:3000/board/n ==> Board 컴포넌트

localhost:3000/login==> NoMatch 컴포넌트

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import {BrowserRouter}
from "react-router-dom";
import App from "./App";

ReactDOM.render(
  <BrowserRouter>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </BrowserRouter>,
  document.getElementById("root")
);
```

App.js

```
import React from 'react';
import { Route } from
'react-router-dom';
import Main from './Main';
import Board from './Board';

const App = () => {
  return (
<Routes>
<Route path='/' element={<Main />} />
<Route path='/boardlist'
element={<BoardList />} />
<Route path="/board/*"
element={<Board seq="1" />} />
<Route path="*" element={<NoMatch />}
/>
</Routes>
  );
};


export default App;
```

Main.js

```
const Main=()=>
(<div><h1>나의 홈페이지</h1>
<img src=
"https://i0.wp.com/adtmoneylab.com
/wp-
content/uploads/2021/07/social-
media-
1177293_1280.png?resize=1024%2C774
&ssl=1">
```

Board.js

```
const Board=(p)=>(<div><h1>게시물
{p.seq}번 </h1></div>)
export default Board;
```

</img>) export default Main;	
BoardList.js-> **hello-react 프로젝트에서 가져오자**	**NoMatch.js** const NoMatch=()=>(찾으시는 url 은 없습니다) export default NoMatch;

final project

조원 그대로

프로젝트 주제 1 차 확장 / 2 차 새로운 주제

기술명 - 프론트엔드 리액트 변환+view 리턴

5/4, 6 일 - 오전 뷰 수업

2 차 - 서비스 배포 x

5/6 월 금요일 - 기획안

5 월 9-20 일 2 주동안 구현/통합

5/23,24 - ppt / 산출물 / 발표

<Main />
<BoardList />
<LoginForm />
<BoardWriterForm />

1. /main url --> <Main />-->index.html

1. /boardlist url --> <BoardList />-->index.html

<Routes>

<Route path="/main" element= {<Main />} />
<Route path="/boardlist" element= {<BoardList />} />
<Route path="*" element="{<NoMatch />}" />

</Routes>

<Link to="/boardlist> 게시판화면 이동 <Link>